Nachdenkzettel Exceptions

Aufgabe 1:

1. Ein Catch-Statement sollte nie leer sein, da sonst (fatale) Fehler gefangen werden und keine Benachrichtigung oder Warnung ausgegeben wird. Dadurch kann das Programm nicht auf den Fehler eingehen und sich dementsprechend anders verhalten.

2. Die Reihenfolge der Catch-Statements ist von Bedeutung. In diesem Fall tritt das zweite Catch nie ein, da auch diese Exception bereits vom ersten Catch gefangen wird. Dementsprechend muss die Reihenfolge der beiden Catch-Statements vertauscht werden.

verbesserter Code:

```
public void doSomething() {
        try {
                look();
        } catch (ConcurrentModificationException e) {
                System.out.println("bad stuff going on today!");
        } catch (Exception e) {
                System.out.println("other bad stuff going on today!");
        } finally {
                return;
        }
}
```

Aufgabe 2:

Ausgabe: BCD

Aufgabe 3:

Ein re-throw einer Exception sollte immer dann implementiert werden, wenn…

- die Exception in der aufrufenden Methode behandelt werden soll
- die genaue Exception dem Benutzer angezeigt werden soll

Aufgabe 4:

Dies ist keine Exception, da die Datenbankabfrage zwar kein Ergebnis bzw. ein leeres ResultSet beim Abfragen der Kunden-ID liefert, dies jedoch nicht automatisch eine Exception wirft. Je nach Weiterverarbeitung des ResultSets kann es u.U. zu einer Exception kommen, wenn z.B. ein bestimmter Index des leeren ResultSets aufgerufen werden soll. Hier kann es dann sinnvoll sein eine eigene Exception zu werfen, um in der aufrufenden Methode dem User mitzuteilen, dass er sich noch registrieren muss.

Aufgabe 5:

Exceptions ermöglichen eine bessere Trennung zwischen dem gewünschten Ablauf des Programmes und möglichem Fehlverhalten. Damit wird der Code deutlich übersichtlicher und die Fehlerbehandlung erleichtert.